EXTENSIBLE MECHANISM FOR EXECUTING SERVER SIDE CODE

TECHNICAL FIELD

[0001] The invention relates to client/server computing and more particularly to an extensible mechanism for processing computing code on the server side.

BACKGROUND OF THE INVENTION

[0002] The client/server environment typically consists of both server and client-side services providing a distributed computing model in which client applications request services from server applications. An example of a client/server system is a business application that allows a user to access information on a central database server. Another example of client/server environment is a web environment that includes a continuously growing number of different applications for web environment support, including web browsers, web servers, launchers, etc. A variety of web resources such as Hyper Text Markup Language (HTML) files, Java Servlet Pages (JSP™), Java™ classes and Servlets™, Enterprise Java Beans™ (EJB), Extensible Markup Language (XML) deployment descriptors makes the task of application development and debugging more complicated and time consuming, especially if more than one developer is involved in a project. An integrated development environment (IDE) is usually used to assist in developing relatively large and complex software products for a client server environment. A developer is usually responsible for developing a piece or module of the final software application. Such a module must be written and debugged prior its implementation into the final product for seamless integration with other modules. Another problem related to testing of the final product is based on the necessity of using a number of client server environment support applications such as web browsers, application servers, etc. Variations between particular client server environment support applications such as the differences between two application servers from different sources makes the task of a developer more complex, because the developer, in addition to engineering compatibility with other modules of the business applicant, has to provide compatibility with a number of different servers and

web browsers. Testing the compatibility of one or more development resources with one or more servers or one or more client side applications can consume a great amount of the developer's time which extends the duration of product development and makes the product more expensive.

[0003] A solution to some or all of these shortcomings is therefore desired.

SUMMARY OF THE INVENTION

[0004] The present invention is directed to an extensible mechanism for execution server side code in a client server environment.

[0005] Therefore, according to an aspect of the invention there is provided a method of executing server side code in a client server environment. The method comprises: processing an input object identifying code for executing on a server, said processing using a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object; processing the deployable object using a server list of at least one server element to determine a server for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object; and processing the launchable object using a launcher list of at least one client element to determine a client for launching the code on the particular server. The method may further comprise launching the client determined in response to the launchable object and executing the code on the particular server.

[0006] In accordance with a feature of the invention, at least one of the view list, server list and launcher list is extensible to accommodate additional respective elements. The method may further comprise maintaining at least one of the view list, server list and launcher list.

[0007] Accordingly, the step of processing the input object may comprise: analyzing the input object to determine an input object element for processing the input object; and processing the input object using the determined input object element. The step of

processing the deployable object may comprise: analyzing the deployable object to determine a server element for processing the deployable object; and processing the deployable object using the determined server element. The step of processing the launchable object may comprise analysing the launchable object to determine a client element for processing the launchable object; and processing the launchable object using the determined client element.

[0008] According to another aspect of the invention there is provided an extensible mechanism for executing server side code in a client server environment comprising: a view mechanism for processing an input object identifying code for executing on a server and outputting a deployable object; a server mechanism for processing the deployable object to determine a particular server for executing the code and to enable the deployable object to execute on the particular server, said second mechanism outputting a launchable object; and a launcher mechanism for processing the launchable object to determine a client for launching the code on the particular server. Accordingly, the view mechanism may comprise a view list of at least one input object element, each input object element processing a type of code identified by the input object for outputting the deployable object. The view list is extensible to accommodate additional respective elements.

[0009] Further, the server mechanism may comprise a server list of at least one server element, each server element enabling the deployable object to execute on a particular server and processing the deployable object for outputting a launchable object. The server list is extensible to accommodate additional respective elements.

[0010] Also the launcher mechanism may comprise a launcher list of at least one client element, each client element enabling the launchable object to execute on a particular client for launching the code on the particular server. The launcher list is extensible to accommodate additional respective elements.

[0011] In accordance with a feature of the invention, the extensible mechanism is adaptable to launch the client determined in response to the launchable object for executing the code on the particular server.

[0012] Also, the extensible mechanism may be adapted to be integrated into an integrated development environment.

[0013] There is provided, in accordance with an aspect of the invention, a computer program product embodied in a computer readable medium for instructing a computer system to perform a method in accordance with the invention herein. Further, there is provided a computer readable medium storing data and instructions readable by a computer system, said computer system executing an integrated development environment (IDE) for generating code for executing in a client server environment, said data and instructions defining an extensible mechanism for executing said code on a server that, when deployed on said computer system, adapts said IDE to: process an input object identifying code for executing on a server, said processing using a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object; process the deployable object using a server list of at least one server element to determine a server for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object; and process the launchable object using a launcher list of at least one client element to determine a client for launching the code on the particular server.

[0014] In accordance with yet a further aspect of the invention, there is a method of maintaining an extensible mechanism for executing server side code in a client server environment. The method comprises maintaining at least one of: a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object; a server list of at least one server element to determine a server for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object; and a launcher list of at least one client element to determine a client for launching the

code on the particular server. In accordance with feature of this aspect, maintaining comprises at least one of: generating a respective element for; adding a respective element to; configuring a respective element of; and deleting a respective element from; at least one of the view list, server list and launcher list. Further, this method may comprise executing server side code using at least one of the view list, server list and launcher list.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] Further features and advantages of the present invention will become apparent from the following detailed description, taken in combination with the appended drawings, in which:

[0016] Fig. 1 schematically illustrates a computer system embodying aspects of the invention;

[0017] Fig. 2 schematically illustrates in greater detail, a portion of the computer system of Fig. 1;

[0018] Fig. 3 illustrates in functional block form, a portion of the memory illustrated in Fig. 2;

[0019] Fig. 4 illustrates a schematic diagram of a system for executing server side code associated with a predetermined type of a web resource.

[0020] FIG. 5 illustrates a schematic block diagram of a portion of the memory of FIG. 3; and

[0021] FIG. 6 illustrates operations in accordance with the invention for processing an input object.

[0022] It will be noted that throughout the appended drawings, like features are identified by like reference numerals.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0023] An embodiment of the invention, computer system **100**, is illustrated in **FIG.1**. Computer system **100**, which is illustrated for exemplary purposes as a computing device, is adapted to communicate with other computing devices (not shown) using network **102**. As will be appreciated by those of ordinary skill in the art, network **102** may be embodied using conventional networking technologies and may include one or more of the following: local networks, wide area networks, intranets, the Internet, and the like.

[0024] Through the description herein, an embodiment of the invention is illustrated with aspects of the invention embodied solely on computer system **100**. As will be appreciated by those of ordinary skill in the art, aspects of the invention may be distributed among one or more networked computing devices which interact with computer system **100**, using one or more networks such as, for example network **102**. However, for ease of understanding, aspects of the invention have been embodied in a single computing device – computer system **100**.

[0025] Computing device **100** typically includes a processing system **104** which is enabled to communicate with the network **102,** various input devices **106**, and output devices **108**. Input devices **106**, (a keyboard and a mouse are shown) may also include a scanner, an imaging system (e.g., a camera, etc.), or the like. Similarly, output devices **108** (only a display is illustrated) may also include printers and the like. Additionally, combination input/output (I/O) devices may also be in communication with processing system **104**. Examples of conventional I/O devices (not shown in **FIG.1**) include removable recordable media (e.g., floppy disk drives, tape drives, CD-ROM drives, DVD-RW drives, etc.), touch screen displays, and the like.

[0026] Exemplary processing system **104** is illustrated in greater detail in **FIG. 2**. As illustrated, processing system **104** includes a number of components: a central processing unit (CPU) **202**; memory **204**; network interface (I/F) **206;** and input-output interface (I/O I/F) **208**. Communication between various components of the processing system **104** may be facilitated via a suitable communications bus **210** as required.

[0027] CPU **202** is a processing unit, such as an Intel Pentium™, IBM PowerPC™, Sun Microsystems UltraSparc™ processor, or the like, suitable for the operations described herein. As will be appreciated by those of ordinary skill in the art, other embodiments of processing system **104** could use alternative CPUs and may include embodiments in which one or more CPUs are employed (not shown). CPU **202** may include various support circuits to enable communication between itself and the other components of processing system **104**.

[0028] Memory **204** includes both volatile memory **212** and persistent memory **214** for the storage of: operational instructions for execution by CPU **202**; data registers; application and thread storage; and the like. Memory **204** preferably includes a combination of random access memory (RAM), read only memory (ROM), and persistent memory such as that provided by a hard disk drive.

[0029] Network I/F **206** enables communication between other computing devices (not shown) and other network computing devices via network **102**. Network I/F **206** may be embodied in one or more conventional communication devices. Examples of a conventional communication device include: an Ethernet card; a token ring card; a modem, or the like. Network I/F **206** may also enable the retrieval or transmission of instructions for execution by CPU **202**, from or to a remote storage media or device via network **102**.

[0030] I/O I/F **208** enables communication between processing system **104** and the various I/O devices **106** and **108**. I/O I/F **208** may include, for example a video card for interfacing with an external display such as output device **108**. Additionally, I/O I/F **208** may enable communication between processing system **104** and a removable media **216**. Removable media **216** may comprise a conventional diskette or other removable memory devices such as Zip™ drives, flash cards, CD-ROMs, static memory devices, and the like. Removable media **216** may be used to provide instructions for execution by CPU **202** or as a removable data storage device.

[0031] The computer instructions/applications stored in memory **204** and executed by CPU **202** (thus adapting the operation of computer system **100** as described herein) are illustrated in functional block form in **FIG. 3**. As will be appreciated by those of ordinary skill in the art, the discrimination between aspects of the applications illustrated as functional blocks in **FIG. 3**, is somewhat arbitrary in that the various operations attributed to a particular application as described herein may, in an_alternative embodiment, be subsumed by another application.

[0032] As illustrated for exemplary purposes only, memory **204** stores a number of applications and data for enabling the operation of system 100 to provide an extensible mechanism for executing server side code; namely: an operating system (OS) **302**, a communication suite **304**, an integrated development environment (IDE) **306**, an extensible mechanism (EM) **308** for executing server side code which plugs into or adapts IDE **306**, and EM lists, collectively **310,** for EM **308** comprising a view list of input object elements **310a**, a server list of server elements **310b** and a launcher list of client elements **310c**.

[0033] OS **302** is an operating system suitable for operation with a selected CPU **202** and the operations described herein. Multi-tasking, multi-threaded OSes such as, for example IBM AIX™, Microsoft Windows, Linux, or the like, are expected to be preferred in many embodiments. Communication suite **304** provides, through interaction with OS **302** and network I/F **206** (**FIG. 2**) suitable communication protocols to enable communication with other networked computing devices via network **110** (**FIG. 1**). Communication suite **304** may include one or more of such protocols such as TCP/IP, Ethernet, token ring and the like.

[0034] IDE **306** is an application for generating code, such as a business application, for running in a client server environment. IDE **306** includes various editors, tools and wizards for, among other things, creating code, deploying code to a server and testing such code. IDE **306** is adapted to provide a user interface such as a graphical user interface (GUI) for such operations as is well known to persons of ordinary skill in the art.

[0035] EM **308** using the EM lists **310**, provides a mechanism for executing server side code on one of one or more servers and in association with one of one or more client applications (i.e. "clients"). In accordance with a model of EM **308**, the execution of server side code is partitioned into three stages, the view stage for determining the code for running, the server stage for determining the server to execute the code and the launcher stage to determine the client for interacting with the server to run the code. EM lists **310** extensibly configure EM **308** to work with different types of server side code to be run (per view list **310a**), different servers to run the code (per server list **310b**) and different clients with which to interact with the server running the code (per launcher list **310c**). The EM lists 310 each comprise one or more respective elements. Each element provides an interface for receiving an input to the element and provides for the processing of the input to produce an output for processing by elements of subsequent stages in the EM model as described further below.

[0036] By modeling the execution of server side code in three stages EM **308** enables the use of different servers and launchers for executing different code types and provides a mechanism to easily add additional or modify exiting servers, launchers and code types to EM **308**. As described further herein below, each of the EM lists **310a, 310b, 310c** of respective input object, server and client elements may be maintained independently of one another and are extensible. For example, a server element for enabling execution of code on a particular server may be added to server list **310b** or an existing element in such list amended without regard to the elements of lists **310a** and **310c**.

[0037] As will be understood by persons skilled in the art, elements in the lists may be implemented as XML documents (or other descriptive language document), Java code objects or other coded objects or modules (e.g. C/C++) or combinations thereof (e.g. a Java code object and XML document). The inputs and outputs therefore may also comprise similar structures such as XML documents, Java code objects or combinations thereof.

[0038] **FIG. 4** illustrates a schematic diagram of an exemplary structure of a client-server environment and EM **308** integrated in IDE **306**. A well known client-server environment is a web environment based on a server side part **402** and a client side part **404** shown schematically separated by a dotted line. The illustrated client-server environment for exemplary purposes is embodied using one computing system **100**, but it must be understood by those of ordinary skill in the art that the environment may be distributed among one or more computing devices which are enabled to interact with each other using a network such as network **102.**

[0039] Server side part **402** typically comprises at least one server (two exemplary HTTP servers **406** and **408** are illustrated) providing an environment on which to run different code. Such code collectively **410** may comprise a variety of different types of resources such as HTML files, JSPs, Java classes and Servlets, EJBs, XML deployment descriptors, graphic files, video, animation and audio files, etc. Three types of resources are shown for illustrative purposes e.g. an HTML file **410a**, a GIF file **410b** and a JPG file **410c**. Client side **404** comprises client side code that interacts with a server running the code to perform some function, for example, access to one or more resources. The client side code may be an application such as an Internet browser, an FTP client, a background process, etc. In the illustrated diagram the client side part **404** comprises exemplary client side code, namely a browser **416** and performance analyzer **418** that are compatible with at least one of servers **406**, **408**. The applications of the server side and client side are enabled to communicate with each other using a standard protocol such as Hyper Text Transfer Protocol, File Transfer Protocol (FTP) or Simple Object Access Protocol (SOAP) over HTTP.

[0040] **FIG. 5** illustrates a schematic block diagram **500** of EM **308**. EM **308** comprises a view module **502**, a server module **504** and a launcher module **506.** Each of the modules **502, 504** and **506** may be implemented as a reusable piece of computer code with specific inputs and outputs. The modules **502, 504** and **506** are preferably adapted to be plugged in and/or removed easily from an application, for example, IDE **306**. Each of the modules **502, 504** and **506** is adapted by a corresponding EM list

310a, 301b and **310c** to handle respective inputs and produce respective outputs for handling by subsequent modules in accordance with the EM model described above.

[0041] In order to describe operations and features of EM **308** and system **100**, the following definitions are used. A deployable object is an output, such as a Java code object, XML document etc., that contains information about code to be run on an as yet unidentified server. Depending on the type of code to be run, the deployable object therefor contains enough information to publish the code to a server (though not necessarily a specific server) and may contain information on what the resource is "contained" in, (i.e. how it is packaged). For example, a deployable object for an HTTP file may contain a filename or other identifier to construct a Universal Resource Locator (URL) for the HTML file to be run on an HTTP server. A launchable object is an output, such as a Java code object, XML document etc., that contains information on how to access code to be run (i.e. a specific resource) on a particular server, and information about how to route access to the resource, (e.g. firewalls and/or the type of client application that can be used to access the web resource) if the access information is not sufficiently clear. For example, a URL may be constructed and provided to direct access to the resource when a Web browser is intended to consume the object. However, a specific resource may require a specific plug-in in a browser that supports the specific resource. For example, an HTML file may have a Macromedia Shockwave™ reference. In order to fully display the HTML page, a browser intended to display the page requires an appropriate Macromedia Shockwave™ plug-in installed. The launchable object for such an HTML page contains information specifying that the HTML file requires a browser having Macromedia Shockwave™ plug-in installed. Also, a launchable object may contain information such as a URL and an IP address and/or port number.

[0042] View module **502** is adapted to receive (e.g. from a user selection via user interface **420)**, an input object **508**, which could be some text in an editor, an object or file in a file manager, an item from a combo box, etc. View module **502** is adapted by view list **310a** of elements for processing input objects which elements are used to identify a particular input object and configure it for execution on a server outputting in response to the input object a deployable object **511** for further handling by server

module **504**. View list **310a,** for exemplary purposes, comprises only three elements for three types of input objects, namely HTML **503a**, GIF **503b** and JPG **503c**. View module **502** is adapted to communicate with user interface **420** in order to notify a user about processing of the input object or to receive user input for an input object such as input specifying a type of resource if view module **502** is unable to recognize the input.

[0043] Server module **504**, is adapted to receive deployable object **511** from view module **502** for further processing to run the code identified by the deployable object on a particular server. Sever module **504** is adapted by server list **310b** to prepare the desired code for execution on at least one of one or more servers enabled by the elements of server list **310b** and to output a launchable object for further processing by launcher module **506.** In **Fig. 4** only two exemplary servers are available to system **100** namely, Apache HTTP server **406** and IBM HTTP server **408**). Server list **310b** is configured with elements **505a** and **505b** respectively enabling each of these servers in the EM model. If the received deployable object **511** is recognized by at least one of the elements **505a** and **505b**, server module **504** adapted by the element outputs a launchable object **512** that provides the information needed to access the deployable object **511** on the particular server **406** or **408.**

[0044] Server module **504** is adapted to communicate with user interface **420** in order to notify the user about the processing of the deployable object **511**, for example to facilitate the selection of a particular server **406** or **408** when more than one element may handle the deployable object or to specify the deployable object **512** as a specific type of deployable object, if server module **504** is unable to automatically recognize the input. If server module **504** cannot recognize the deployable object **511**, a launchable object **512** is not output.

[0045] Launcher module **506**, is adapted to receive launchable object **512**, and launch an appropriate client on the client side **404** for that particular type of launchable object **512**. Launcher module **506** is adapted by launcher list **310c** of client elements which enable various clients. Exemplary launcher list **310c** comprises a Web browser element **507a** and a performance analyzer element **507b** for respectively launching web

browser **416** or performance analyzer **418** in response to appropriate launchable objects. Launcher module **504** is adapted to communicate with user interface **420** in order to notify the user about processing of the launchable object **512**, or to specify the launchable object **512** as a specific type of launchable object, if the launcher module **506** is unable to recognize the input.

[0046] **FIG. 6** illustrates a flow chart **600** of the main steps of operations for executing server side code associated with a type of resource in accordance with an aspect of the invention.

[0047] After beginning operation, (step **602**) an input object is selected, for example, by a user using the user interface **420**. Such an interface may be an IDE interface for developing code to be run on one or more types of servers in communication with one or more client side applications. Persons of ordinary skill in the art will appreciate that code under development often requires testing in a client-server environment and that the present invention may be useful to facilitate such testing.

[0048] The selected object **508** could be some text in an editor, an object or a file in a file manager, an item from a combo box, etc. Operations identify input object **506** (step **604**), using view list **310a** of elements for handling input objects. If input object **508** is not recognized, e.g. characteristics of input object **508** do not match at least one type of input object for which view module is pre-configured using view list **310a**, the input object cannot be configured for running on a server and execution of the operations is ended (step **606**). If input object **508** is recognized and can be associated with only one type of web resource included in view list **310a**, input object **508** is processed (step **608**) as defined by the element of view list **310a** and a deployable object **511** is output (step **610**).

[0049] If input object **508** is recognized but is associated with more than one element of view list **310a**, the user ay be prompted by a notifying message via user interface **420** to select the element that is to handle the input object **508** for further processing (step **608**). On receipt of the selection from the user interface **420**, a

deployable object **511** is output (step **610**). If user interface **420** does not specify a type of input, input object **508** cannot be run and thus execution of the method is ended (step **606**).

[0050] On receipt of the deployable object **510**, server module **504** adapted by elements of server list 310b identifies a server that can be used to run the received deployable object (step **614**). If there are no servers that can be used, deployable object **511** cannot be run and the operations are ended (step **606**). If there is only one server identified as compatible with deployable object **511**, the identified server is started (step **616**) in order to run deployable object **511**. The code to be run may be published to the server if so enabled by the element. A launchable object **512** (step **618**) is output to launcher module **506**. If more than one server is identified as being compatible with the deployable object, server module **504** displays the list of available servers to the user and allows the user to select one server (step **620**). On receipt of the user's selection, processing proceeds as described for an identified server. If user interface **420** does not specify a type of server, input object **508** cannot be run and thus execution of operations is ended (step **606**).

[0051] On receipt of launchable object **512**, launcher module **506** identifies client side code ("clients") that can be used for interacting with the server running the code (step **622**). If there are no clients that can be used, launchable object **512** cannot be run and execution is ended (step **606**). If only one client is identified as compatible with launchable object **512**, the identified client is started (step **624**) in order to run the launchable object **512**. If more than one client is identified as compatible with the launchable object, launcher module **506** displays the list of available clients to the user and allows the user to select one (step **626**). On receipt of the user's selection, the selected client is started in order to run launchable object **512**. If no client is identified, processing ends (step **606**).

[0052] As a result of the extensible nature of the EM model, an element may be added to one of the EM lists **310** for which there may not be a suitable element in one or more of the other lists. For example, a new server element may be added to server list

310b for which no suitable client element yet exists in launcher list **310c.** In order to support enhanced service for a user and avoid a situation wherein a choice is presented at one stage and then on the next stage the use is notified that the choice cannot be competed, in one embodiment of the invention, EM **308** is configured to pre-check the compatibility of inputs (e.g. input object **508**, deployable object **511** and launchable object **512)** with each corresponding module (e.g. view module **502**, server module **504** and launcher module **506**) prior the actual processing of input object and displaying dialogs to the user. The system programmatically traces through possible outcomes of the steps to ensure that each step will be allowed to go all the way through. If a given input cannot go through at least each of one of the view module, server module or launcher module, the system does not process the input and will notify the user that the selection cannot be processed.

[0053] Persons skilled in the art will appreciate that numerous modifications to the above described embodiment are possible while remaining within the scope of the invention. For example, the exemplary embodiment describes a user interface for selecting code to be run and resolving the determination of particular elements to process inputs at stages of the EM model. However, other embodiments may include predetermined preferences specified earlier by the user for determining among different elements or may employ operations for automatically choosing a "best" element or autonomic programming techniques. The list of elements may be ordered whereby the first element matching the input is used.

[0054] EM lists **310** may be maintained or modified in a variety of manners such as through IDE **308** adapted to edit the lists. IDE wizards or tools may be provided to configure new elements or edit old ones, providing an interface to receive certain parameters and descriptors and generate appropriate elements to EM lists **310**. Such elements may be created independently without IDE assistance and imported or combined with EM lists **310**. An element of an EM list may be created or configured manually using a simple text editor or a special editor such as one adapted for editting XML. A tool or other application for creating or configuring a view, server or launcher to be enabled by the present invention may be adapted to generate some or part of an

element for one of the EM lists **310**. How each of EM lists **310** is stored is not important, per se. For example, one of EM lists **310**, such as view list **310A**, could be an XML file. The other EM lists **310** could be Java code. Importantly to support extensibility, at least some of EM lists can be modified, configured, or extended.

[0055] Extensibility is advantageous for a number of reasons. A small change to just one of EM lists **310** (e.g. to add a new launcher) can open up a new path for launching a different resource type. When each of EM lists **310** is populated, adding a single element to server list **310B**, for example, might provide full support for running and launching all resources on that specific server type, even with very little new code or development for whoever created the server element. As such, the present invention may facilitate less development and testing time and less complexity through modularity (adding a new server type does not require the developer to know or re-implement views and launchers). Extending one of EM lists **310** may provide additional benefits beyond those originally contemplated. For example, a developer may add a view for a particular server and in doing so, provide the view to other servers automatically.

[0056] A further embodiment may permit EM lists **310** to be modified during the runtime of EM **308**. For example, a user could be prompted when selecting a new resource type that doesn't work, and they can select, create, or configure a new server/launcher/etc. to handle the input.

[0057] The embodiment(s) of the invention described above is(are) intended to be exemplary only. The scope of the invention is therefore intended to be limited solely by the scope of the appended claims.